monitored network. DIDS can potentially handle hosts without monitors since the LAN Monitor can report on the network activities of such hosts. The Host and LAN Monitors are primarily responsible for the collection of evidence of unauthorized or suspicious activity, while the DIDS Director is primarily responsible for its evaluation.

Reports are sent independently and asynchronously from the Host and LAN Monitors to the DIDS Director through a communications infrastructure (Fig. 10). The architecture also provides for bi-directional communication between the DIDS Director and any monitor in the configuration. This communication consists primarily of notable events and anomaly reports from the monitors. The director can also make requests for more detailed information from the distributed monitors via a "GET" directive, and issue commands to have the distributed monitors modify their monitoring capabilities via a "SET" directive. A large amount of low level filtering and some analysis is performed by the Host Monitor to minimize the use of network bandwidth in passing evidence to the director.

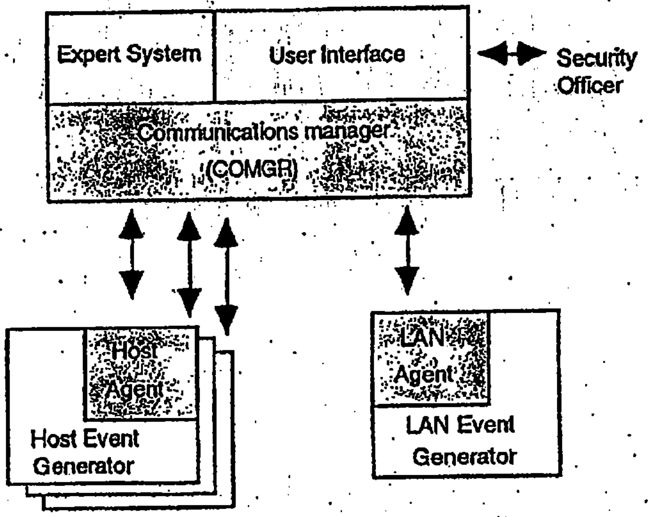


Figure 10. Communications Architecture

The Host Monitor consists of a Host Event Generator (HEG) and a Host Agent. The HEG collects and analyzes audit records from the host's operating system. The audit records are scanned for notable events, which are transactions that are of interest independent of any other records. These include, among others, failed events, user authentications, changes to the security state of the system, and any network access such as rlogin and rsh. These notable events are then sent to the director for further analysis. The HEG also tracks user sessions and reports anomalous behavior aggregated over time through user/group profiles and integrates Haystack [20] reports into DIDS. The Host Agent handles all communications between the Host Monitor and the DIDS Director.

Like the Host Monitor, the LAN Monitor consists of a LAN Event Generator (LEG) and a LAN Agent. The LEG is currently a subset of U.C. Davis' NSM [8]. Its main responsibility is to observe all of the traffic on its segment of the LAN in order to monitor host-to-host connections, services used, and volume of traffic. The LAN Monitor reports on such network activity as rlogin and telnet connections, the use of security-related services, and changes in network traffic patterns. The LAN Monitor is especially helpful in monitoring network activity from hosts without Host Monitors.

The DIDS Director consists of three major components that are all located on the same dedicated workstation. Because the components are logically independent processes, they could be distributed as well. The Communications Manager is responsible for the transfer of data between the director and each of the Host and the LAN Monitors. It accepts the notable event records from each of the Host and LAN Monitors and sends them to the Expert System. On behalf of the Expert System or User Interface, it is also able to send requests to the Host and LAN Monitors for more information regarding a particular subject. The Expert System is responsible for evaluating and reporting on the security state of the monitored system. It receives the reports from the Host and the LAN Monitors, and, based on these reports, it makes inferences about the security of each individual host, as well as the system as a whole. The Director's User Interface allows the System Security

Officer (SSO) interactive access to the entire system. The SSO is able to watch activities on each host, watch network traffic, and request more specific types of information from the Monitors.

Chapter 4

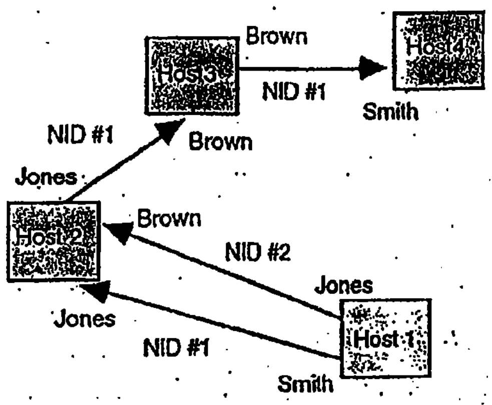
The DIDS Expert System

DIDS utilizes a rule-based expert system which is divided into two logical units. The first tracks users movement through the network by the generation of NID's. And the second uses rules to attach intrusive and suspicious activity to the aforementioned NID's.

4.1 Distributed Recognition and Accountability

With respect to Unix, the only legitimate way to create an alias of a user is for the user to login from a terminal, console, or off-LAN source, to change the user-id, or to create additional aliases (local or remote) for an existing user. In each case, there is only one initial login (network wide) from an external device and a new unique NID is created when this original login is detected. When a user spawns a new login session (alias), that new session is associated with his original NID. Every subsequent action generated by that user or his aliases is then accounted to that NID. Thus, the system performs DRA to maintain a single identification for each physical user (see Figure 11).

Note that if a user creates an alias by a non-legitimate method which goes undetected by the auditing system, DIDS will alert the SSO as soon as it observes activity which cannot be linked to an NID. From here on, it is assumed that aliases are created in one of the above legitimate manners.



Order of Events

Smith@host1 -> Jones@host2
Jones@host1 -> Brown@host2
Jones@host2 -> Brown@host3
Brown@host3 -> Smith@host4

Figure 11. NID's

When a user creates a new alias of herself within a local host, the local audit trail is sufficient to document the aliasing. However, when a user creates a remote alias using rlogin, ftp, etc.; the local audit trail only documents half of the aliasing; the other half of the aliasing is on the remote host's audit trail. Since collecting all audit trails and processing them centrally is impractical in a large network, DIDS performs pre-processing of the audit trails at each end of a network connection and subsequently sends summary reports to the Expert System. These summary reports will be discussed in the next section.

In contrast, the rlogin protocol under Sun Microsystem's BSM package.[20] automatically requests the user-id from the source host prior to completing each network connection. This permits the remote host's audit trail to retain the original user-id from the originating host. Thus the original user-id is passed from host to host as the user traverses the network. Although the BSM approach is conceptually cleaner, it is not feasible unless both hosts agree to the protocol modification, i.e., both are running SunOS 4.1 and BSM. On the other hand, our approach works in a heterogeneous environment without modification to existing protocols. As proof, we are extending DIDS to Digital's VMS operating system in our next release.

4.1.1. The Algorithm

Figure 12 is our model of a user creating a remote alias via a network connection. Our DRA approach is to monitor every network connection between hosts, and trace all connections back to the originating user ID. As mentioned, there are several programs involved: audit demons on both source and destination machines, preprocessors and Host Monitors on both source and destination machines, a LAN Monitor and the Expert System.

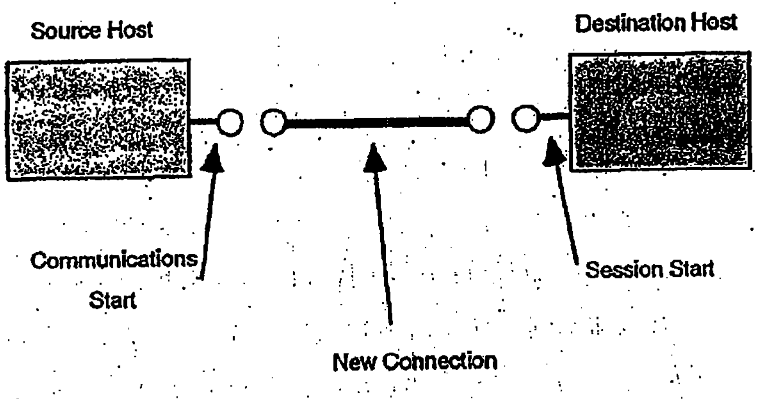


Figure 12. A Network Connection

When a user creates a remote user alias via connection; (in no specific order),

1. If the host at the source of the network connection is monitored, the corresponding Host Monitor notifies the Expert System via a Connection Start (CS) packet that the user at the source host is attempting a network connection

2. If the host at the destination end of the network connection is monitored then this Host Monitor notifies the Expert System via a Session Start (SS) packet as to whether the user has successfully created a remote alias or failed
3. The LAN Monitor detects the new connection and notifies the Expert System of both host ids and also a connection number via a New Connection (NC) packet.

Each of the three packets mentioned above contain: (source_host_name, source_port_number, destination_host_name, destination_port_number), which we call the conn_tuple. It should be noted that Unix port naming conventions allow a port to have only a single owner, therefore connections can be uniquely defined (at a specific time) by the conn_tuple.

The rules used to unite the information from the CS packet, SS packet and NC packet are the following.

1. If we have an SS_1 which shares the conn_tuple with an existing connection_item, then replace the SS_2 information in the connection_item if SS_1 is newer. If no such connection_item exists then create a new one, and fill it with the values contained in SS_1.

2. If we have an NC_1 which shares the conn_tuple with an existing connection_item, then replace the NC_2 information in the connection_item if NC_1 is newer.

3. If we have a CS_1 which shares the conn_tuple with an existing $connection\_item_1$, then replace the CS_2 information in the $connection\_item_1$ if CS_1 is newer.

3'. If there is a $connection\_item_2$ whose destination_host_name and destination_uid match the source_host_name and source_uid of $connection\_item_1$, then associate the NID of connection_item2 with connection_item1 (in section 4.1.3 this rule will be expanded).

The reason that a check for the existence of "old" information in a connection_item is necessary is that packets signaling the close of a connection may be delayed or lost during network transmission. It should also be noted that time values associated with the SS, NC and CS within a connection_item are initially -1, which allows for the first CS and NC to be filled into a connection_item.

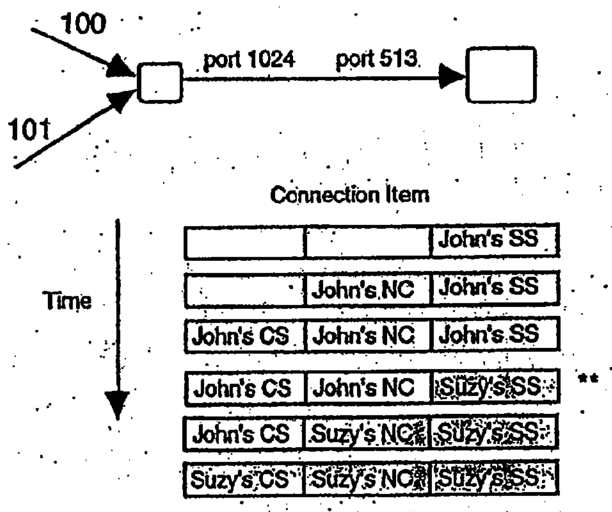


Figure 13. Foiling an Intruder

Figure 13 shows graphically how this algorithm may work in practice. Assume first that user John has logged into host1, as has user Suzy. Also assume that they have NID's #100 and #101 respectively. Now, user John telnets to host2 using the account name of smith, and successfully logs in. This connection generates the CS, NC and SS packets which arrive in no specific order, but are processed SS first. Now, Suzy who wishes to conceal her identity on host2 attempts to steal the ports used by John (NID #100). To do this, she kills the telnet process which is running on host1, which closes John's connection, but does not alert host2 or the DIDS Expert System, because such a closure is not picked up by the auditing system. Since Unix recycles its ports Suzy can now connect to host2 using the same ports as John used (and assume she does so using the smith account name and a password she stole from John).

The DIDS Expert System however realizes that a new connection has been started (and the old one stopped) because it has SS, CS and NC packets which overwrite a previous connection. At this point, by the second part of rule 3 above, NID#101 is associated with Suzy's new connection, thus foiling her attempt to cover her tracks. As Figure 13 shows (see the ** marked step), Suzy's SS packet is for a short time associated with John's CS and NC, which would cause Suzy's activity to be associated with John until her CS packet arrives at the Expert System. This is a problem that is inherent in this DRA implementation because at no point can we conclude that a connection finally has the correct combination of packets. On the other hand this algorithm *does* guarantee that given *all* of the packets (CS and SS) for a given conn_tuple, it will resolve correctly (assuming both hosts are monitored). This guarantee has so far proved sufficient in practice. The next section gives a proof of this algorithm, and section 4.1.3 presents an extension to this algorithm which has yet to be implemented, but does guarantee that all activity will be linked to the correct user (e.g., any activity by Suzy will be linked to her).

4.1.2 Proof of Algorithm

In order to prove that connection_i between two monitored hosts can be correctly resolved given all the CS and SS packets for connection_i's conn_tuple, we must enumerate the possible cases.

1. CS and SS are received by the Expert System in the order in which they are generated.
2. Local logins. That is to say, logins from a host to itself.
3. Late CS packet. This describes the situation when connections may continue out of the destination_host of connection$_i$ before the CS packet for connection$_i$ is received.
4. Late SS packet. Similar to case 3, but the SS packet is the one that is delayed.
5. Multiple matching NID's previous to connection$_i$ (see Figure 14).

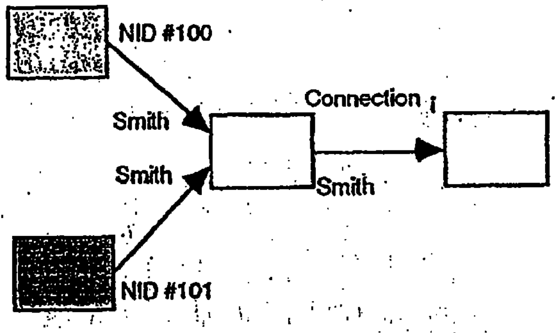


Figure 14. Multiple NID's Before

6. Multiple matching NID's following connection$_i$ (see Figure 15).

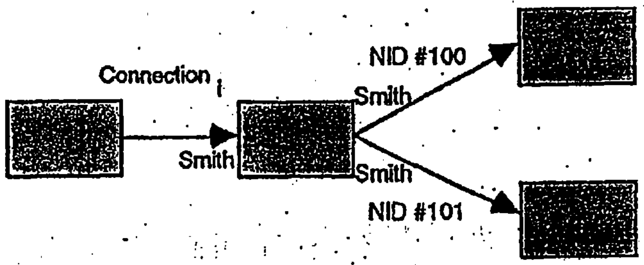


Figure 15. Multiple NID's After

Our experience has shown case 1 to be the most likely case. Since the packets arrive in order, it is easy to put the CS and SS packets together to form a connection_item. By following the algorithm stated above, if there are multiple CS and SS packets (for a particular conn_tuple), only the latest will be used to form the connection. It is obvious that since we assume the arrival of all CS and SS packets, and the fact that time stamps increase monotonically, the correct combination of CS and SS packets will be used to form the connection.

Case 2 actually reduces to cases 1, 3, 4, 5 or 6 because although these connections don't go out onto the network, they are still assigned a pair of ports just like in case 1.

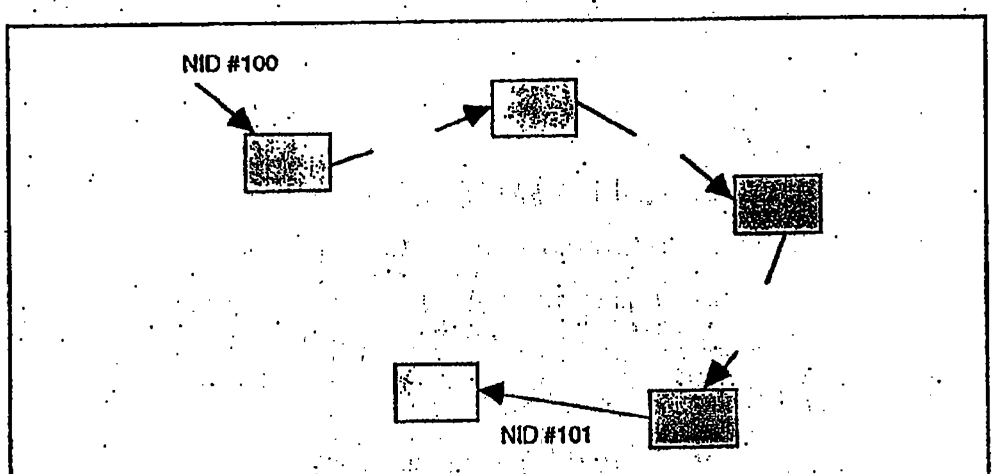


Figure 16. Chain of Incomplete Connections

Cases 3 and 4 can be handled in exactly the same way. That is, if we view a connection as the *pair* of CS and SS, it doesn't matter which is late. Figure 16 show the general state which occurs when either a CS or SS is late. As the figure shows, within a chain of connections that should be connected to NID #100, there may be connections that have not been formed yet because of a missing CS or SS. These connections (by algorithm above) are assigned *new* NID numbers. In order to handle this case correctly, we must augment step 3 of the algorithm:

3. If we have a CS_1 which shares the conn_tuple with an existing $connection\_item_1$, then replace the CS_2 information in the $connection\_item_1$ if CS_1 is newer. (Of the 3 sub-rules below, the last has precedence.)

3'. If there is a $connection\_item_2$ whose destination_host_name and destination_uid match the source_host_name and source_uid of $connection\_item_1$ (condition 1), then associate the NID of $connection\_item_2$ with $connection\_item_1$.

3". If there is a $connection\_item_3$ whose source_host_name and source_uid match the destination_host_name and destination_uid of $connection\_item_1$ (condition 2), then associate the NID of $connection\_item_2$ with $connection\_item_1$.

3'". If condition 1 and condition 2 hold, then associate the NID of $connection\_item_2$ with $connection\_item_1$. In addition, change all of the connections which have the same NID as $connection\_item_3$ to the NID which $connection\_item_1$ now has.

By using this new rule we can associate the whole chain of connections shown in figure 16 with the same NID, which is the desired result.

Case 5 (see Figure 14) is handled by associating $connection\_item_i$ with *both* NID's. In this way, we are guaranteed to have a trail back along both NID's (since we only distinguish by user-id on a particular host) if activity out of $connection\_item_i$ turns out to be intrusive.. In the next section, we will use TTY's in order to continue tracking NID's even when two NID's share the same user-id on a host.

Case 6 (see Figure 15) can occur because of cases 3 and 4. That is, two or more connections out of a particular host could be created before $connection\_item_i$ is finally formed. Each of those connections would be assigned a new and different NID (from the rules). To handle this case, $connection\_item_i$ and all of the matching connections out of $connection\_item_i$'s destination host, are assigned the *same* NID. This results in the correct associations.

The above cases demonstrate that the DRA algorithm utilized by DIDS does indeed track users as they move across the monitored network.

4.1.3 Extensions to the DRA Algorithm

The previous section showed that given all CS and SS packets, the DRA algorithm presented guarantees that if a human user makes a connection, then it will be linked to him by an NID. As mentioned above, this algorithm has proved sufficient in all situations encountered by DIDS so far. But there remains the possibility that activity of an intrusive user will be associated with the wrong NID, as was shown in Figure 12. In order to insure that all activity will be associated with the proper user (and no one else), addition infotmation is required. First, sequence numbers must be associated with CS and SS packets so that the Expert System can be sure that it has connected the proper pair. Second, CS and SS packets must include the source and destination TTY of the connection respectively. And last, all reports of user activity must include the pid included in the session start audit record of that particular session.

To prove that these extensions allow us to prove that *all* activity of a user will be associated with that user and only that user, we will follow the proof above.

The proof of case 1 (proper ordering of CS and SS) and 2 (local logins) can clearly be covered as special cases of cases 3-6.

Case 3 (late CS) differs from the proof of the previous case 3 in that we must keep track of the activity which is associated with the corresponding SS packet which has already arrived. Since the SS packet contains a sequence number greater than any CS, we can conclude that a new CS is pending. Therefore, any activity reports which are marked with the pid of the SS can be held until the CS arrives. This insures that this activity will not be associated with the wrong user. Since we assume that the CS for this connection will eventually arrive bearing a sequence number matching the SS, we can then associate any waiting activity to the source side of the connection (and the corresponding NID).

Case 4 (late SS) is similar to case 3. If activity reports arrive at the Expert System which are tagged with a pid that is not associated with an existing SS, then hold it until the SS arrives. And again hold the CS with the higher sequence number than any existing SS. Once the SS matching the CS for the current connection arrives, then attach the SS and any associated activity reports to the corresponding CS (and NID).

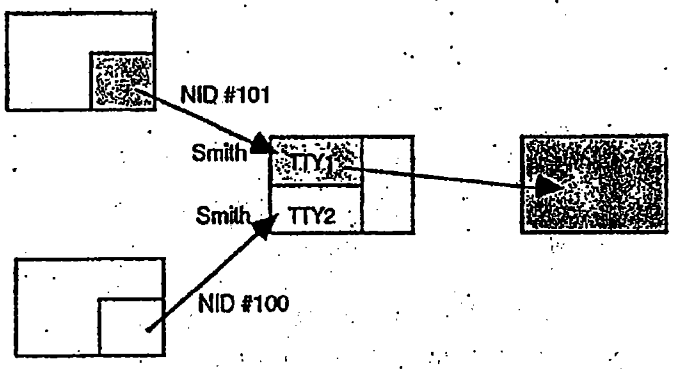


Figure 17. Multiple NID's before

Case 5 relies on the use of TTY's (see Figure 15). Since each incoming connection which shares a particular user-id enters at one and only one TTY, and since connection$_i$ can originate from one and only one TTY, it is trivial to link the activity on connection$_i$ with the appropriate connection.

Case 6 mirrors the reasoning of case 5. By the use of TTY's, it is obvious that activity can be linked correctly.

The above cases demonstrate that this extension of the implemented DRA algorithm guarantees that activity will be correctly associated with its instigator.

4.1.4 Special DRA Cases

DRA can continue to work to some extent when either the source or the destination machine is unmonitored. In the case of logins from off the monitored network, it is possible to link the activities back to that unmonitored host using the source_host_name provided in the SS packet from the monitored destination machine. Likewise, it is possible to follow an NID out connection to an unmonitored machine using the information in a CS packet. But a problem lies in that in neither of these cases can we determine the user identity on the unmonitored host. This problem suggests a way that an attacker could cover his tracks. By simply logging into an unmonitored host then back onto the monitored network, an attacker can effectively cover his tracks (see Figure 11). This is where the LAN Monitor can provide some much needed help.

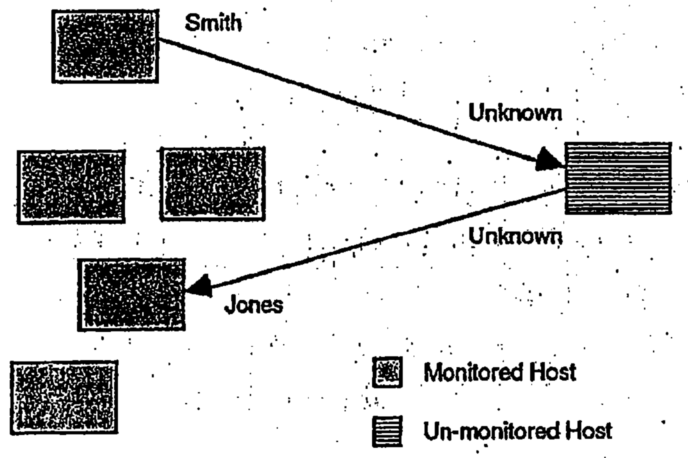


Figure 18. Off then Back On.

A "thumbprint" is an abstraction used by the LAN Monitor which represents data flow along a network connection for a specified time period. By comparing the "thumbprints" of the outbound and inbound connections, it is possible to determine if the two connections belong to the same user [9]. This is because any activity by a user on the furthest out connection must pass through each of the other connections on the path back to the user's original login. Therefore, whenever the Expert System sees a connection from an unmonitored host, it can request that the LAN Monitor compare it to all connections which leave the monitored network, thus extending the DRA capabilities of DIDS.

4.2 Detecting Intrusive Behavior

Figure 15 shows the IDM (intrusion detection model) which provides structure to the DIDS Expert System [1]. It shows the conceptual flow of low level events as they are processed and affect the security states of NID's, hosts (monitored and un-monitored), and the network itself.

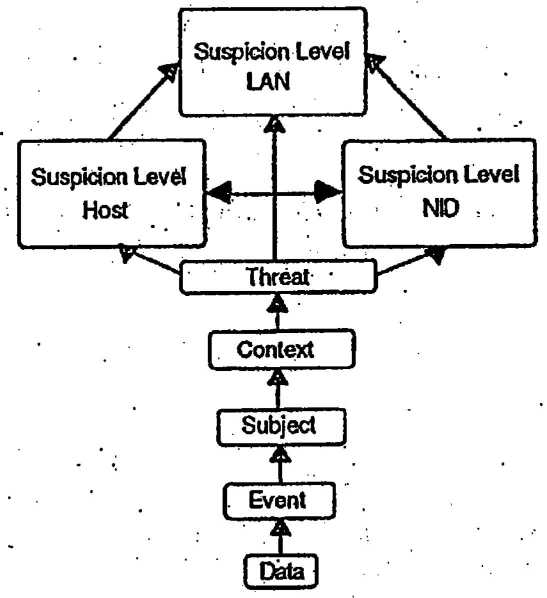


Figure 19. Intrusion Detection Model (IDM)

The abstraction at each level are:

1. Audit records provided by the host operating system, by the LAN monitor, or by a third party auditing package,
2. Events (which have already been discussed in the context of the Host and LAN Monitor),

3. A subject which is a single identification for a user across many hosts on the network, and which identifies by NID,
4. Event in Context (events are placed in context with other events),
5. Threats posed by NID's on particular hosts or the network itself where the threat representation is an short English description of what the Expert System perceives particular NID(s) are doing that is of a suspicious nature, and
6. Security state of NID's, hosts and the network as functions of all the threats perceived.

Upper levels of the model treat the network-user as a single entity; essentially ignoring the local identification on each host. Similarly at the highest level, the collection of hosts on the LAN are generally treated as a single entity. The security states are numeric values between 1 and 100 which provide a quick reference point for the SSO. All the evidence which is used in the calculation of the security state is retained for further reference by the SSO.

4.2.1 Events

This section presents the events which are currently used by the DIDS Expert System to detect intrusive behavior.

1. Connection_items. These are the fundamental unit of DRA representing a login from one host to another (may be the same).

2. Failed Connections. These are the same as above, but the user failed to enter the correct password.

3. Suspicious Session Reports. These reports come from the Host Monitor and signal some form of suspicious activity. The suspicious activity currently reported include; browsing, paranoia, leakage, malicious use, security penetration and mobility. For a further description of these reports see [20].

4. Tagged File accesses. These reports come from the Host Monitor and alert the Expert System that a sensitive file has been accessed. It is possible to tag files for read, write or execute accesses separately. This is important because for instance, reads to the password file are numerous and not suspicious, while writes are indeed suspicious.

5. Tagged Port Access. These reports come from the Host Monitor and alert the Expert System that a suspicious port has received a network connection.

6. String Matches. These reports come from the LAN Monitor and signal the Expert System to possible suspicious activity. For instance the string "passwd" may suggest that a passwd file is being transferred over the network.

7. Host signatures. These reports come from the Host Monitor and signal that a particular known attack may be occurring. See [21] for a further description of these signatures.

8. LAN Anomalies. These reports are sent by the LAN Monitor and reports the LAN Monitor's perceived level of suspicion about a particular connection. The items that the LAN Monitoring considers in forming this anomaly value include load on connection, service utilized, etc. For further information about these anomaly values see [8].

4.2.2 Events in Context (Suspicious Activities)

Below is the set of "events in context" which represent the suspicious activities which DIDS utilizes in the formation of a NID's security state.

1. NID Loops. This refers to a chain of a chain of at least two connections followed by a trivial loop connection. This behavior has been witnessed to be used by masqueraders in an attempt to cover their tracks. This is accomplished in unmonitored environments because when the legitimate user logs in, the "Last Login:" message says that it was from that same machine.

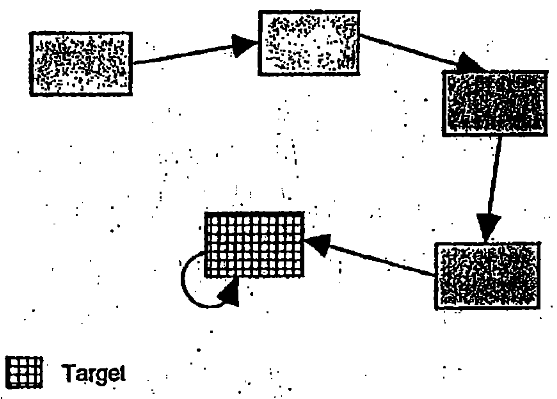


Figure 20. NID Loop

2. Doorknob Attack. This is a very common attack that consists of trying several user-id/password combinations, hoping to find an account with either an easy password or no password (see item 9 below).

3. Multiple NID per User-id. This occurs when two or more NID's converge on the same host using the same uid. This activity signifies that a masquerader and a legitimate user may be on at the same time. In this case, since we don't know for sure who is who, all involved NID's are considered guilty.

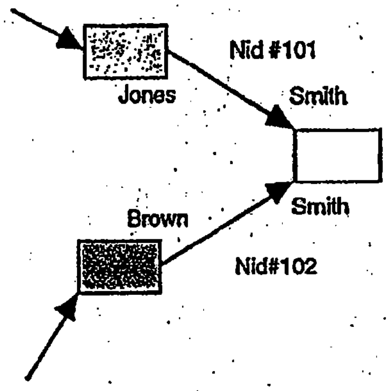


Figure 21. Multiple NID's

4. UID Switches. This is when someone either makes a new connection which whose source user-id differs form the destination user-id, or when a command like "su" is used to change your identity. Switches to users with higher privileges is extremely suspicious.

5. Chaining. This is a series of connections which extend outward from the initial connection. It is thought to be very suspicious to go out more than a single hop. This activity is often used by attackers to hide their true identity.

6. Normal Hosts. The DIDS Expert System keeps track of two sets of hosts per user: first, a set of hosts that are legitimate originating connections, and second, a set of legitimate destination hosts.

7. Normal Aliases. This is a set of user account names that a particular user has legitimate access to.

8. Normal Time. This is a set of time spans which the user is expected to log in during, perhaps set to 9-5 Monday-Friday.

9. Default Accounts. This is a set of default accounts which exist on many systems. These accounts are generally one of the objects of attack by a "cracker" because the passwords are often nonexistent or set to their default value.

10. Off_Then_On Connections. This is a pair of connections that the LAN Monitor has discover are rooted at the same initial login. One of the connections leaves the monitored network, and the other returns back to it. This is to guard against "insiders" hiding their tracks (as discussed in section 4.1.4).

11. From_off Connections. In some computing environments, it will be useful to treat all connections that originate off the monitored network as suspicious.

12. To_off Connections. As with item 11 above, on some sites it may be suspicious to leave the monitored domain.

13. Connections to or from Sensitive Hosts. Any connection to a host of a different security level may be suspicious. Connections to less secure hosts may signal leakage of sensitive information, while connection to a more sensitive host may signal a masquerader.

Currently the above set of NID attributes is used in a weighted sum to form the aggregate warning level for a particular NID. Some of the above attributes also go into the host and LAN warning levels such as doorknobs and mobility (which measures then number of users who are logged into a particular host or LAN). Host and LAN warning levels also inherit the suspiciousness of the users on the system. That is to say, if a suspicious NID logs into a host, then that host becomes suspicious.

In order to describe how these contexts can be used to form the "Threats" level of the IDM, it is necessary to introduce the machine learning techniques which are used to form those "Threats." In the next section, we discuss extensions to traditional decision trees which allow them to utilize *hierarchical attributes* as well as *stepwise attribute introduction and deletion.*

Chapter 5

DIDS and Machine Learning

Applying machine learning to the intrusion detection task will allow for the automatic generation of expert system rules as well as a scheme for assigning suspicion levels to users. In addition, using decision trees with *stepwise feature introduction and deletion* will provide a easy way to extend the Expert System to allow for new sources of information, new attack forms, and site specific policies.

Section 5.1 discusses some empirical results which show that introducing new features to a decision tree can be more effective than starting training from scratch. Also, Section 5.2 shows *hierarchical attributes* to be an effective tool at reducing decision tree error rates, and how this type of attribute can be utilized in intrusion detection.

5.1 Stepwise Feature Introduction and Deletion

Traditional decision tree algorithms assume that all the attributes used in the inductive learning task are known before training begins. But, this is certainly not the case with IDS's. With new system vulnerabilities being discovered all the time, and the presence of site specific security policies, it is obvious that the set of user and system attributes utilized by an IDS must change in order to keep up with the task. It may turn out that some attributes which are currently being used do not provide any useful information about the "intrusiveness" of a particular user. And it may turn out that new attributes are required or at least useful in decreasing the number of false positives and/or false negatives.

One benefit of using decision trees to help develop an expert system is that they automatically form a set of easily understood classification rules. This allows the human user to "tweak" the classifier by introducing hand picked training examples, in order to correct malformed rules as Shapiro's Interactive ID3 is intended to do [18]. But Shapiro's work is limited in that it only allow human experts to select new training examples that might improve the quality of the decision tree. If the attributes are insufficient for the classification task, then no number of additional examples no matter how carefully selected, can improve the error rate past a certain point. What may be needed is a new attribute.

The next section shows that new attributes indeed can be added to an existing training set without having to start collecting training instances from scratch.

5.1.1 Empirical Results

The goal of this study was to compare decision trees which use *stepwise feature introduction* to plain ID3 which starts training from scratch every time a new attribute is added. The general approach is to fill in values for attributes which were not present in the training instances when they were collected (because the attribute was unknown). In order to determine the best method for filling in missing values, three different methods were tested.

The first is the fraction-example method which assigns a fraction of the examples with missing values into subsets based on the proportion of the examples which have a particular value. In addition, the information gain of an attribute is reduced by the proportion of missing values to known values. The second method of filling in values is to simply fill in the most common value, given the class of the example (before information gain is calculated). These two methods were shown to work well in the general "missing value" problem investigated by Quinlan [15]. The last method pertains only to attributes with continuous values, which consists of just filling in the average of the known values.

The first set of tests dealt with three well known machine learning dataset which had only categorical attributes. These datasets were:

1. Faulty LED. This is a training set that consists of seven binary attributes, each corresponding to an element of a LED display. Each element has a 10% probability of being on when it should be off or vice versa. There were 200 training examples, and 5000 testing examples.
2. Chess. This set consists of 75 training examples and 125 testing examples whose 36 attributes describe various relationships between chess pieces. The goal is to determine if white can win given a particular setting of 4 remaining chess pieces. These pieces are the white king and rook, and the black king and pawn.
3. Soybean. This training set consists of examples which contain a total of 35 attributes (some not very useful) and 19 different classes of soy bean diseases. There were 150 training examples and 150 testing examples.

Table 2 shows the results of the first test. The attributes were divided into three sets, each consisting of a randomly selected third of the available attributes. Training began with 1/3 of the total training instances using only the first third of the attributes (the rest being unknown). In the next phase, the second 1/3 of the training instances were added (using the first 2/3 of the training instances). The last phase consisted of adding the remaining training instances, each containing the complete set of attributes. Also included in Table 2, are the results of starting training from scratch with the last 1/3 of the training instances.

Method of Replacement	LED	Chess	Soy Bean
Most Common	33.0%	8.3%	53.3%
Fraction Example	34.9%	17.8%	27.6%
Start From Scratch on last 1/3	50.2%	11.1%	81.0%

Table 2. Average case with Attributes added in Thirds

Tables 3 and 4 show the results of the best and worst case tests. Instead of randomly dividing up the attributes, they were instead divided up according to the information gain heuristic described above. In the best case the attributes are added in descending order (1/3 at a time), and in the worst case the attributes were added in ascending order (1/3 at a time).

Method of Replacement	LED	Chess	Soy Bean
Most Common	33.0%	0.0%	51.3%
Fraction Example	32.9%	0.0%	30.0%

Table 3. Best Case with Attributes added in Thirds

Method of Replacement	LED	Chess	Soy Bean
Most Common	32.1%	13.0%	45.3%
Fraction Example	39.4%	29.2%	41.0%

Table 4. Worst Case with Attributes added in Thirds

The data in Tables 2, 3 and 4 suggest that the best performing method of replacement (most common or fraction example) is data specific. But clearly, both of these methods are superior to starting training from scratch every time a new attribute is introduced.

The second experiment that was done was to start training with 1/3 of the attributes, then add the rest of the attributes after some amount of training. The purpose of this test is to determine the effects of training initially with either the very best or worst of a set of attributes, then continuing training with instances that include all the attributes. Figures 22 23 and 24 show the average case error rates for different numbers of training examples with complete sets of attributes (rest have only the first third). Also included is data for training with full attribute vectors only. This provides a performance baseline which can be used to gauge the success of the fill-in methods. When the fill-in methods do worse than the baseline, then obviously incorrect values are being filled into the training instances causing additional error.

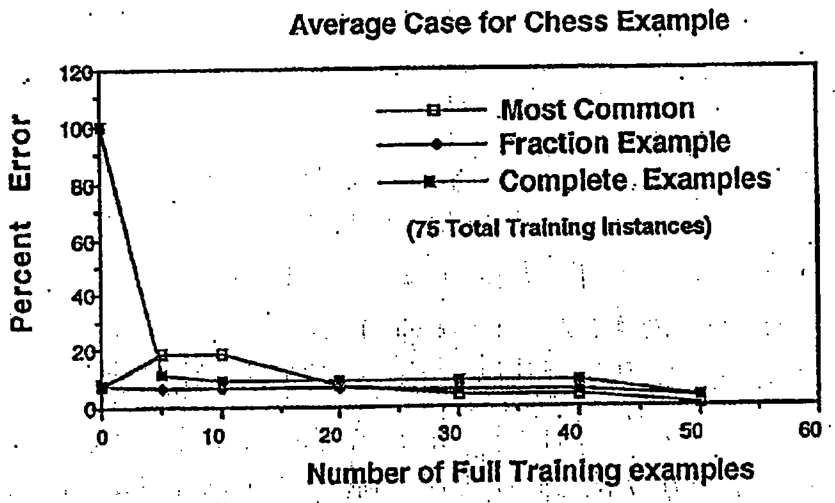


Figure 22

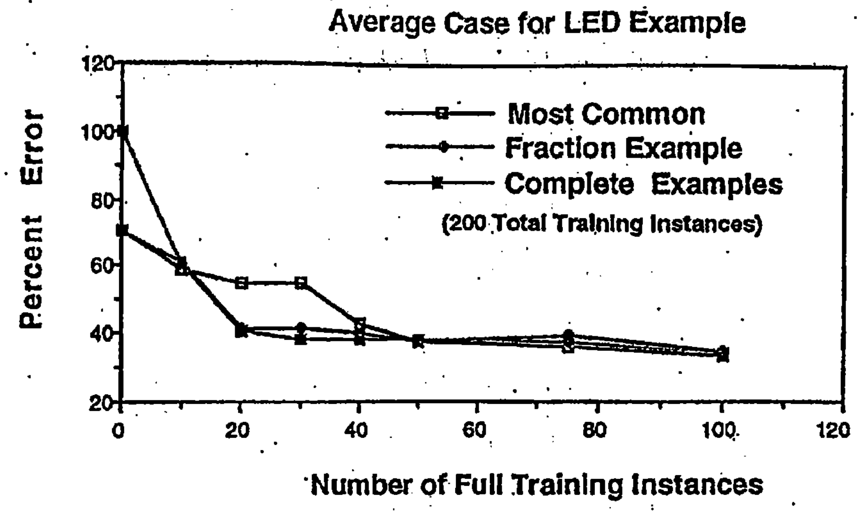


Figure 23

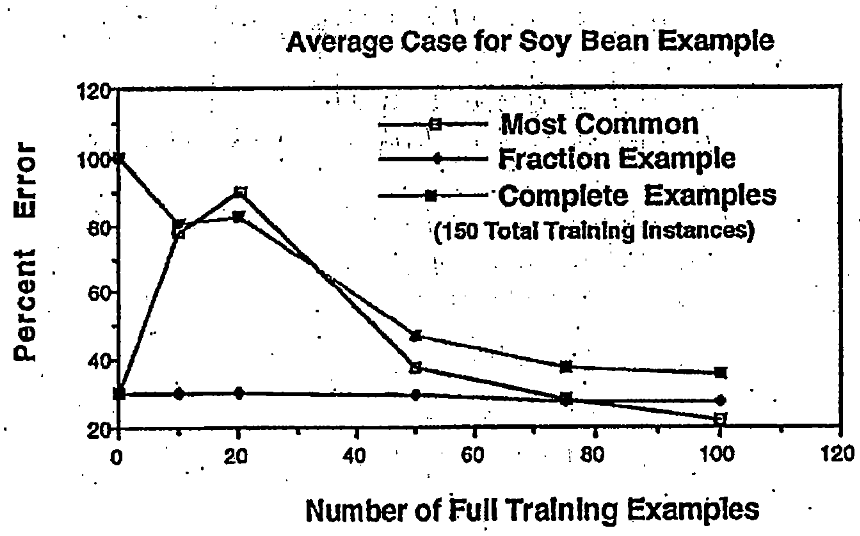


Figure 24

Figures 23, 24, and 25 clearly demonstrate that the "most common" replacement method does poorly when the number of missing values is very high. Actually, in all three training sets "most common" replacement does worse than starting training from scratch. But as the number of complete training instances increases past 50%, the "most common" replacement method appears to catch up and even surpass the performance of the "fraction-example" method. The other result demonstrated from the above graphs is that it requires between 10-70 (complete) training instances to reach asymptotic lower bound on error. This number clearly is data dependent, but suggests that starting training from scratch can be costly.

In order to test fill-in methods on attributes with continuous attributes, a further set of tests were done. The two methods tested were fraction-example and the average value method. Table 5 and Figure 25 show the results of the above experiments done on the Iris database (classification of Iris flowers) which consists of 4 continuous attributes, 75 training examples and 75 testing examples.

Method of Replacement	Best	Average	Worst
Average Values	5.8%	7.7%	8.3%
Fraction Example	5.7%	8.2%	21.9%

Table 5. Iris Data Added in Thirds

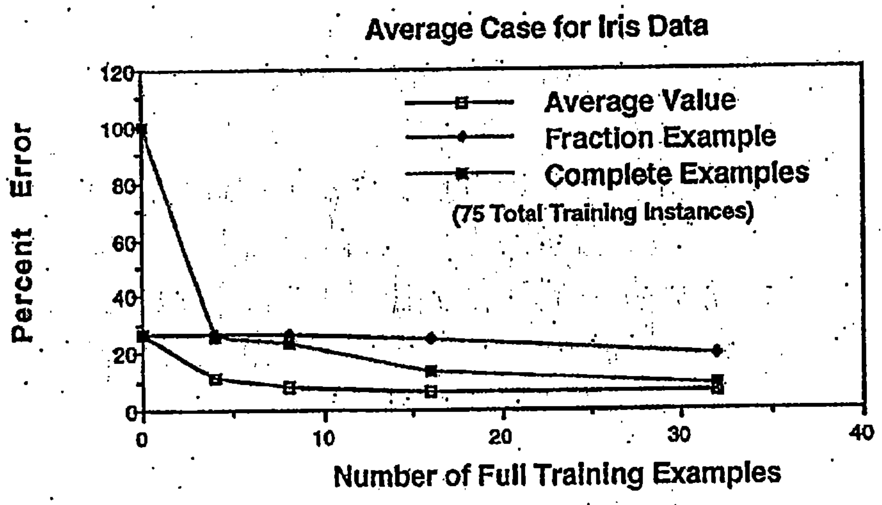


Figure 25.

Replacing continuous attributes' missing values with the average of the known values appears to do very well (as shown in the above figures). The fraction-example actually performed worse than restarting training from scratch, which clearly suggests that this method should not be used.

The results of all the above experiments show that it is clearly better to incrementally add attributes, then it is to start training from scratch, or continue training with a poor set of attributes. The other result that was discovered was that overall the fraction-example method did not do as well as (relatively) as was suggested by the results found by Quinlan [16] (especially when the number of missing values is low). This is most likely due to the structured nature of the attribute fill-ins that occurs in these experiments as opposed to the random missing values which were used in his experiments.

Fraction-example did out perform the other methods when the initial set of attributes is very good, and the number of examples with "complete" attribute vectors is small. This is understandable since the fraction-example method reduces the information gain of attributes with missing values. This would keep the poorer attributes (which had a large proportion of missing values) down toward the bottom of the tree, where damage (to classifying ability) is kept to a minimum.

Overall, the above results suggest that when adding a new attribute (with categorical values) it is best to use the fraction-example method in building the decision tree until a significant number of training examples with values for that attribute have been acquired. The number of such training instances will clearly be data dependent but the results above suggest that 30-50 training instances should prove sufficient. Once this group of augmented training instances has been acquired, then switch to the "most common" replacement method since it has been shown to work better.

5.2 Hierarchical Attributes

Another addition to traditional decision trees that may prove helpful in intrusion detection, is the use of hierarchical attributes. Concepts such as directory structures, computer networks and privilege groups are concepts that may better be described as a hierarchy of values than some set of flat values (see Figure 26). This is especially true when attributes may take a large number of possible values, because ID3 has been shown to work poorly with such attributes [17].

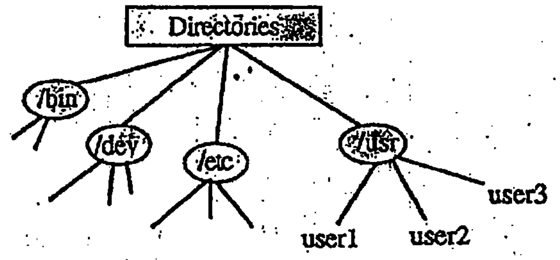


Figure 26. Directory Hierarchy

In order to utilize hierarchical attributes, it was necessary to modify the method by which the ID3 algorithm selects which attribute to split on.

1. Use the information gain heuristic to choose the best attribute utilizing only the highest level values for each attribute. Assume attribute A_1 is chosen.
2. Attempt to build the subtrees under attribute A_1. If the trees can be built with each leaf containing only one class (no noise), then stop. If not, then assume the value of A_1 under which the noise is present is V_1. Now repeat 1, but this time use the children values of V_1 instead of V_1.

In order to test the effectiveness of hierarchical attributes, we will use a simple card example. The training examples have two attributes each: color and value. Figure 27 shows their respective hierarchies and Figure 28 shows a sample tree.

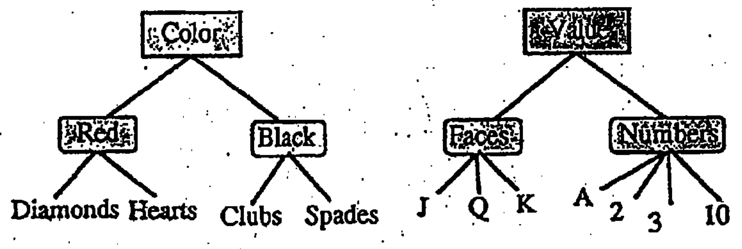
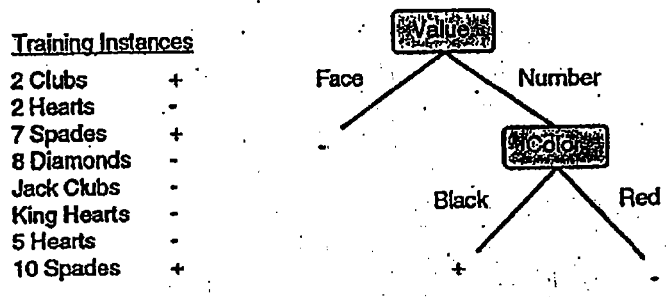


Figure 27. Cards Attributes



2 Clubs	+
2 Hearts	-
7 Spades	+
8 Diamonds	-
Jack Clubs	-
King Hearts	-
5 Hearts	-
10 Spades	+

Figure 28. Examples: Black Numbers (with hierarchical attributes)

As Figure 28 shows, the two attributes can take on their high level values (Face, Number, Red and Black) and still correctly classify the training instances. This decision tree represents the concept in a much more "human comprehensible" manner than would the tree generated by traditional ID3 (see Figure 29).

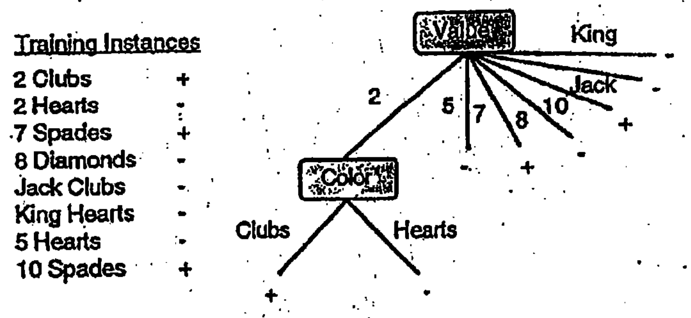


Figure 29. Example: Black Numbers (Plain ID3)

In order to test the general effectiveness of ID3 with hierarchical attributes, a variety of concepts were used. These concepts ranged from general concepts to very specific disjunctive concepts. Training and testing sets consisted of a set of single cards drawn at random from a deck (no duplication). Table 6 shows the results using 1/2 of the deck as training data and the other 1/2 as the testing data. Table 7 shows the results when the training set was only 1/3 of the deck, and the testing set was the other 2/3.

Concept	Traditional ID3	Hierarchical Attributes
Black 3's	2.7 %	0.4 %
Black Numbers	33.8%	0.0%
Cards with Value 10	12.3%	6.2%
Red Aces, 2 of Spades, Red 3's, 4 of Clubs, 5 of Hearts	16.2%	15.4%
3 of Spades, 4 of Clubs, 5 of Hearts, 6 of Diamonds	7.2%	15.4%

Table 6. 1/2 Train 1/2 Test

Concept	Traditional ID3	Hierarchical Attributes
Black 3's	4.6 %	3.4 %
Black Numbers	37.8%	0.0%
Cards with Value 10	17.1%	4.6%
Red Aces, 2 of Spades, Red 3's, 4 of Clubs, 5 of Hearts	16.3%	21.1%
3 of Spades, 4 of Clubs, 5 of Hearts, 6 of Diamonds	8.1%	26.2%

Table 7. 1/3 Train 2/3 Test

Overall the hierarchical attributes outperform the regular ID3 algorithm by a large margin. The last two test concepts however show that the hierarchical attributes tend to over generalize (see Tables 6 and 7). This is because the algorithm for building these decision trees with hierarchical attributes will generalize as long as there is no evidence which is contrary to this generalization. Therefore when concepts are very specific and the training set is small, there is a fair chance that the concept learned will be overly general. While in this cards example, the over generalization causes higher error rates, in IDS's over generalization may be the proper action to take. For example, assume that we have the hierarchical attribute shown in Figure 30. Also assume that we have seen a single attack come out of a host1 at MIT. We may very well want to generalize that all activity from outside UC Davis is suspicious.

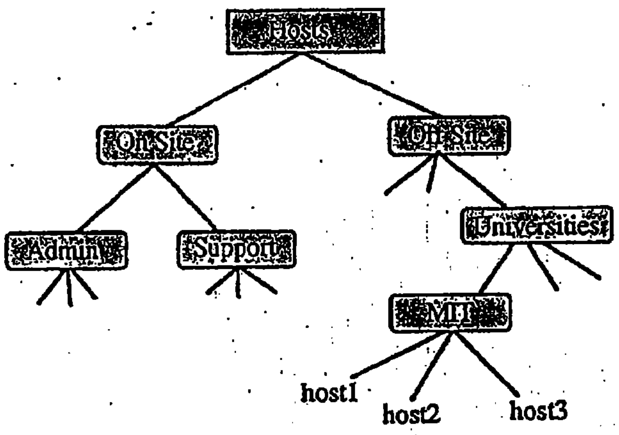


Figure 30. Network Hierarchy

In addition, by introducing another variable per hierarchical attribute, it would be possible to regulate under what conditions the attribute should be generalized. For instance, generalize if two or more of the children have been instantiated with training examples of the same class. Going back to the example in Figure 30, under the above constraint, the low level host1 value would be used until another attack came from another host at MIT, in which case we generalize to "hosts at MIT are bad." Or if an attack originates from another university, then we would generalize to "hosts at universities are bad."

5.3 Utilizing Machine Learning Within DIDS

The primary goals of the DIDS Expert System are to track users as they move across the network, and to detect suspicious activity by cordoning off normal behavior with "tripwires." Examples of such tripwires were given in section 4.2.2, and can be used to partition off "normal" user activities. That is, by keeping a list of machines, I/O devices, directories, programs, user accounts, connection patterns etc. which a user normally uses, we can label any activity which involves other resources or patterns as suspicious. For example, the DIDS Expert System currently keeps a list of hosts that a particular user logs into, and figure 31 shows how unusual logins may be grouped into various suspicion

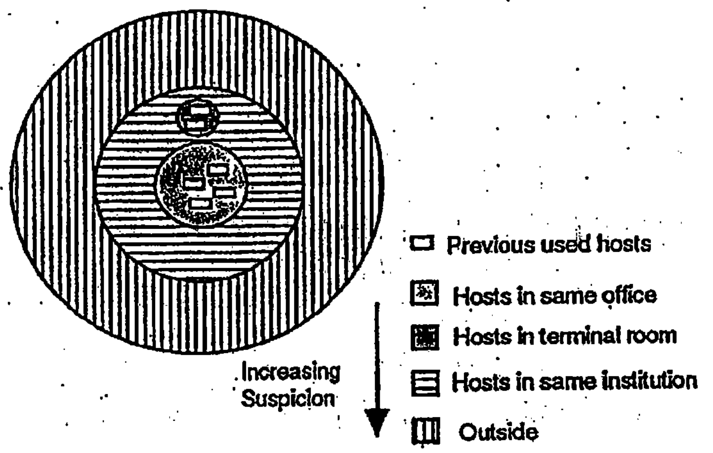


Figure 31. Host Suspicion Levels

levels. The evidence in the previous section suggests hierarchical decision tree attributes can be used to automatically set up these suspicion levels. The other key use of machine learning will be in the deployment of DIDS into widely varying computing environments. It is obvious that a IDS which is "trained" at a university will likely perform less than perfectly at a military installation for instance. Next, we present an example of how the aforementioned machine learning techniques can be utilized in the deployment and tuning of DIDS.

Using a simple weighted sum of the suspicious events shown in section 4.2.2, DIDS will assign suspicion levels to users and hosts. These warning levels will provide guidance to the SSO, in tracking down intruders. When the SSO determines the actual class (e.g., legitimate user or some class of intruder like "masquerader" or "doorknob rattler") of a particular NID, then he can direct the Expert System to save this NID's activity (and class he assigned it) for later use in a decision tree. Actually, the SSO may want to attribute more than one class to the instance if multiple attack forms are present.

If the addition of a new training instance introduces noise into the decision tree (two or more identical training instances have been classified differently by the SSO), then the SSO can review these training instances and determine if a "new" attribute is required to distinguish between attackers and legitimate users, or between types of attack. If so, then new collection facilities can be added to the Expert System, LAN Monitor, and or Host Monitor in order to collect the necessary information which forms the new attribute. It may also be helpful to change the level of generality of the hierarchical attributes if they are found to be over-general or too specific. Now, by the results in section 5.1, training may continue as before (with the addition attribute included).

During the training process two warning levels are associated with each NID and host: the weighted sum warning level mentioned above, and a warning level based on the developing decision tree. This decision tree warning level is the ratio of previously seen users/hosts (who shared the same attribute vector) who were classified as suspicious. In addition to a warning level, the decision tree can also suggest what type of suspicious activity a particular user is engaged in (if any). This description of activity is based on the classes which training instances are assigned.

As the number of training instances increases, the predictive quality of the decision tree based warning level will overtake that of the weighted sum warning level. At this point, the SSO may choose to base his decision (as to whether he should investigate a NID's activity) on the decision tree warning level. The SSO may continue to tune and adapt (for new attacks) the Expert System through the addition of new training data and/or new attributes.

Chapter 6

Conclusion

Through the use of DRA, DIDS greatly expands the capabilities of intrusion detection systems. DIDS can track intruders back to his initial login as well as detect distributed attacks which would go undetected by other IDS's. We proved that the implemented DRA algorithm will attribute all connections made by a user to that user even when he attempts to hide his identity. We also suggested an extension to the implemented DRA algorithm which was proved to associate all of a user's activity (connections as well as any audited activity on the individual hosts used) to him and only him.

Section 5.1 showed that stepwise attribute addition can be used to improve the accuracy of decision trees. It was demonstrated that this method is far superior to starting training from scratch every time a new attribute is introduced, or continuing training with a poor set of attributes. We also showed that hierarchical attributes can be used to improve the accuracy of decision tree classifiers as well as be used to control generality.

Finally in section 5.3 it was demonstrated how decision trees which allow incremental introduction of attributes and hierarchical attributes might be used to improve IDS's. We showed how a set of "tripwires" which partition legitimate activity (without the use of sophisticated statistical algorithms) can be used to detect attacks. And we demonstrate a scheme for deployment and tuning of IDS's which based on empirical results in other domains, may improve their accuracy and flexibility.

Chapter 7

Future Work

Several extensions of the DIDS Expert System in are the works. The first of these is to implement the extended version of DRA which was presented in section 4.1.3. This will provide additional assurance that intrusive activity will be correctly attributed to the corresponding attacker. As part of this extension to DRA, we would like to automate the "thumb printing" mechanism in the LAN Monitor. That is, when the LAN Monitor is at a low level of activity, it can compare connections which leave or enter the monitored network, looking for connections which belong to the same user.

Another extension which is in the works involves scaling DIDS to a WAN (wide area network). This will entail using a distributed expert system to track users and attribute suspicious activity to users. Several separate expert systems would work on LAN's which make up the WAN. They would exchange information when a user's moves (via a network connection) from one LAN to another, thus tracking their activity.

The last extension being investigated involves the introduction of an interactive machine learning system into the DIDS User Interface. The goal of this unit would be to make it easier for an SSO to form training instances, and discover when new attributes are needed. In addition this extension will be used to test the viability of decision trees which allow incremental introduction of attributes and the use of hierarchical attributes in intrusion detection.

References

1. J. Brentano, "An Expert System for Detecting Attacks on Distributed Computer Systems," Masters' Thesis, Dept. of Computer Science, UC Davis, 1991.

2. J. Brentano et al., "DIDS (Distributed Intrusion Detection System) Motivation, Architecture, and an Early Prototype," *Proceedings of the 14th National Computer Security Conference*, Washington, D.C., October 1991, pp. 167-176.

3. D. Denning, "An Intrusion Detection Model," *IEEE Transactions on Software Engineering*, February 1987.

4. P. J. Denning, ed. *Computers Under Attack: Intruders, Worms, and Viruses.* New York: ACM Press, 1990.

5. Department of Defense, *Trusted Computer System Evaluation Criteria*, National Computer Security Center, DOD 5200.28STD, Dec. 1985.

6. G. V. Dias, K.N. Levitt, and B. Mukherjee, "Modeling Attacks on Computer Systems: Evaluating Vulnerabilities and Forming a Basis for Attack Detection," Technical Report CSE-90-41, University of California, Davis, Jul. 1990.

7. D. H. Fisher, "An Empirical Comparison of ID3 and Back-Propagation," *Machine Learning*, vol. 2, pp. 139-172, 1987.

8. L. T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," *Proc. 1990 Symposium on Research in Security and Privacy*, pp. 296-304, Oakland, CA, May 1990.

9. L. T. Heberlein et al., "Internet Security Monitor: An Intrusion Detection System for Large Scale Networks," Proceedings of the 15th National Security Conference, To be published, 1992.

10. L. Hyafil & R.L. Rivest, "Constructing Optimal Binary Decision Trees is NP-complete," *Information Processing Letters*, vol. 5, no. 1, 1976.

11. B. Landreth, *Out of the Inner Circle, A Hacker's Guide to Computer Security*, Microsoft Press, Bellevue, WA, 1985.

12. T. Lunt, et al., "IDES A Progress Report," *Proceedings of the Sixth Computer Security Applications Conference*, Tucson, AZ, December 1990.

13. R. Mooney, "An Experimental Comparison of Symbolic and Connectionist Learning Algorithms," *IJCAI*, pp 775-780, 1989.

14. J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.

15. J. R. Quinlan, "An Empirical Comparison of Genetic and Decision-Tree Classifiers," *Proceedings of Fifth International Workshop on Machine Learning*, pp. 135-141, 1988.

16. J. R. Quinlan, "Unknown Attribute Values in Induction," *Proceedings of Sixth International Workshop on Machine Learning*, A.M. Segre, Ed. Los Altos, CA: Morgan Kaufman, 1989.